

网络同步

Listen-Server和 Dedicated-Server

在UE中，Listen-Server和 Dedicated-Server的区别主要体现在以下几个方面：

- Listen-Server是指一个客户端同时充当服务器和玩家的角色，其他客户端连接到这个主机来进行网络同步。Dedicated-Server是指一个单独的服务器，没有玩家的角色，所有客户端都通过连接服务器来进行网络同步。
- Listen-Server更适合局域网联机，而Dedicated-Server更适合网游。
- Listen-Server和Dedicated-Server在开发流程上没有什么区别，在打包方式上有所区别。其中Dedicated-Server在打包中需要使用源码编译的UE版本，创建一个专用的构建目标文件，并在打包时选择开发服务器模式，会去掉图形界面等冗余部分，提高效率。
- Listen-Server和Dedicated-Server在运行时有不同的角色（Role）和远程角色（Remote Role）。Listen-Server的主机拥有权威（Authority）角色，其他客户端拥有自主代理（Autonomous Proxy）或模拟代理（Simulated Proxy）角色。Dedicated-Server拥有权威角色，所有客户端拥有自主代理或模拟代理角色。
- Listen-Server和 Dedicated-Server在网络复制（Replication）的方式上有不同。Listen-Server只能从服务器向客户端复制Actor和属性，而不能从客户端向服务器复制。Dedicated-Server可以从服务器向客户端复制Actor和属性，也可以从客户端向服务器复制远程过程调用（RPC）。

Actor复制

好的，我尽量详细一点。UE5 的actor复制是指服务器将actor的属性和函数同步到客户端，使客户端能够保持对actor的近似状态。actor复制有两种主要方式：属性复制和RPC（远程过程调用）。属性复制是指服务器在检测到属性值发生变化时，自动将更新发送给客户端。RPC是指服务器在执行某个函数时，将其调用发送给客户端。

为了设置属性复制，你需要做以下几件事（更多见下文Actor和ActorComponent）：

在定义属性的actor类的头文件中，你需要确保在UPROPERTY声明中有replicated关键字作为参数。

在actor类的实现中，你需要实现 `GetLifetimeReplicatedProps` 函数。

在actor的构造函数中，确保 `bReplicates` 标志设置为true。

为了设置RPC，你需要做以下几件事：

在定义函数的actor类的头文件中，你需要确保在UFUNCTION声明中有 `NetMulticast`、`Server` 或 `Client` 关键字作为参数。

在actor类的实现中，你需要在函数名前加上相应的前缀（`NetMulticast`、`Server`或`_Client`）。

在调用RPC时，你需要根据函数的类型（可靠或不可靠）和角色（服务器或客户端）来判断是否需要检查 `HasAuthority` 或 `IsNetMode`。

除了设置属性复制和RPC之外，你还需要考虑一些其他因素，比如actor的相关性、优先级、更新频率等，来优化网络性能和带宽使用。

在listen server下，actor的数据同步遵循以下原则：

- 服务器上的actor拥有权威（Authority）角色，客户端上的actor拥有自主代理（Autonomous Proxy）或模拟代理（Simulated Proxy）角色。
- 服务器上的actor可以通过Replicated标签和GetLifetimeReplicatedProps函数来指定哪些属性需要同步到客户端。
- 服务器上的actor可以通过RepNotify函数来监听属性的变化，并在客户端执行相应的逻辑。
 - 蓝图开启步骤
 - 在变量的Details面板中，将Replication类型设置为RepNotify。
 - 在MyBlueprint窗口中，双击自动生成的On Rep函数，编写客户端执行的逻辑。
 - 在服务器上修改变量的值，触发On Rep函数的调用。
 - cpp启用
 - 对于需要复制的变量标记UPROPERTY(ReplicatedUsing = 自定义OnRep函数名)
- 服务器上的actor可以通过远程过程调用（RPC）来向客户端发送消息或请求。
 - 在函数的UFUNCTION宏中，添加Server、Client或NetMulticast关键字，表示函数在哪里调用和执行。
 - 如果需要，添加Reliable关键字，表示函数必须在远程机器上执行。
 - 如果需要，添加WithValidation关键字，表示函数需要一个验证函数来检查参数是否合法。
 - 在函数体中编写逻辑，并在需要时调用该函数。
- 客户端上的actor不能直接修改服务器上的actor的属性，只能通过RPC来请求服务器修改。
- 客户端上的actor可以修改自己的属性，但不会同步到服务器或其他客户端。
- 优化网络同步的性能的方法有以下几种：
 - 减少不必要的网络同步属性和函数，避免频繁或大量的数据传输。
 - 使用合适的同步条件和可靠性，根据不同情况选择不同的同步方式。
 - 使用PushModel特性，让服务器只有在有变化时才发送数据，减少CPU开销。
 - PushModel特性是UE4.25版本引入的一种网络同步优化方法。它的原理是让开发者在修改同步属性的值时，主动标记该属性已经更改，从而省去服务器对属性的比较检查，减少CPU开销。要使用PushModel特性，需要满足以下条件：
 - PushModel特性只适用于同步属性，不适用于同步函数或事件。并且同步属性必须有Replicated标签。
 - 同步属性必须在GetLifetimeReplicatedProps函数中使用DOREPLIFETIME_WITH_PARAMS_FAST或者DOREPLIFETIME_WITH_PARAMS宏，并设置bIsPushBased为true。
 - PushModel特性需要开发者在修改同步属性的值时，必须使用MARK_PROPERTY_DIRTY或者相关的宏来标记属性已更改，否则可能导致数据不一致或同步失败。

- PushModel特性目前还处于实验阶段，可能存在一些未知的问题或bug，需要谨慎使用。
- 使用压缩或打包技术，减少网络带宽占用。使用压缩或打包技术的方法有以下几种：
 - 使用项目模板或迁移工具来创建项目，避免引入不必要的内容或代码。
 - 在打包设置中，启用在发行打包过程中压缩文件（Compress files during shipping packaging）选项，让UE4在打包过程中对文件进行压缩。
 - 在打包设置中，启用使用Pak文件（Use Pak File）选项，让UE4将项目的资产打包为单个文件或单个包。
 - 在打包设置中，启用生成文件块（Generate Chunks）选项，让UE4生成可用于流式安装的.pak文件块。
 - 在打包设置中，启用编译HTTP文件块安装数据（Build Http Chunk Install Data）选项，让UE4为HTTP块安装文件生成数据。

ENetRole

RemoteRole

RemoteRole 和 Role 的区别在于，RemoteRole 表示 actor 在远程连接中的复制模式，而 Role 表示 actor 在本地的控制权。这种区别产生的原因是因为服务器和客户端对于同一个 actor 的角色可能是相反的。例如，如果服务器上有一个 actor，它的 Role 是 `ROLE_Authority`，表示服务器拥有它的控制权，并且它的 RemoteRole 是 `ROLE_SimulatedProxy`，表示它会被复制到客户端，并在客户端进行模拟。那么客户端上看到的这个 actor，它的 Role 就是 `ROLE_SimulatedProxy`，表示它只是接收服务器的更新，并且它的 RemoteRole 就是 `ROLE_Authority`，表示服务器拥有它的控制权。

角色蓝图 AutoPossessPlayer 问题

多人模式时，应设置默认 Pawn 蓝图的 Auto Possess Player 属性 disabled，否则在玩家人数多于 auto possess 序号时会出现无法控制的问题，例如当选择 Auto Possess Player 0 或者 1 时，如果有两个玩家，那么将有一个 PIE 实例无法控制角色，如果选择 0 或者 1 或者 2 并且有三个玩家时，同样将有一个 PIE 实例无法控制角色

为了让每个玩家都能控制角色，最好将默认 Pawn 蓝图的 Auto Possess Player 属性禁用，而不是选择 0、1 或 2 等序号，因为这样可能会导致多人模式下有的玩家无法控制角色。

不直接调用 Overlap 而是通过网络复制变量来制作的原因

你不能简单地直接调用 overlap 函数来显示，因为这样可能会导致不同客户端之间的不一致。如果你只在客户端调用 overlap 函数，那么只有触发了 overlap 事件的客户端才会显示提示，而其他客户端则不会。**如果你只在服务器调用 overlap 函数，那么只有服务器才会显示提示，而所有客户端则不会。（教程中只在 Server 上启用了碰撞）**如果你在服务器和客户端都调用 overlap 函数，那么可能会出现重复或冲突的提示，而且还会增加网络开销。因此，通过网络同步复制属性的方式来制作，可以保证所有客户端和服务器都能看到一致的提示，而且只需要一次网络传输。

只在服务端启用碰撞事件的原因

因为这是C/S模型（所有逻辑应由服务端处理以保证服务端的权威性）

服务器作为游戏主机，保留一个真实 **授权** 的游戏状态。换句话说，服务器是多人游戏实际发生的地方。客户端会远程控制其在服务器上各自拥有的 **Pawn**，发送过程调用以使其执行游戏操作。但服务器不会将视觉效果直接流送至客户端显示器。服务器会将游戏状态信息 **复制** 到各客户端，告知应存在的Actor、此类Actor的行为，以及不同变量应拥有的值。然后各客户端使用此信息，对服务器上正在发生的情况进行高度模拟。

来自虚幻官方文档

- overlap 事件是用来检测两个物体是否相交的事件，如果在多个客户端上启用，可能会导致不一致的结果，因为不同的客户端可能有不同的延迟或者同步问题。
- overlap 事件通常会触发一些游戏逻辑，比如伤害计算，物品拾取，触发器等。这些逻辑应该由 server 来决定和执行，以保证游戏的公平性和一致性。
- overlap 事件在客户端上启用可能会造成不必要的网络开销，因为客户端需要把 overlap 事件发送给 server，然后 server 再把结果广播给其他客户端。这样会增加网络流量和延迟，影响游戏性能和体验。

本地游戏	网络游戏
	
<p>玩家1按下输入以发射武器。</p> <p>玩家1的Pawn将发射其当前武器以响应此操作。玩家1的武器生成发射物，并播放附带音效和视觉效果。</p>	<p>玩家1在本地机器上按下输入以发射武器。</p> <p>玩家1的本地Pawn将武器发射命令传送给服务器上对应的Pawn。玩家1在服务器上的武器生成发射物。服务器告知所有连接的客户端各自生成玩家1发射物的副本。玩家1在服务器上的武器告知所有客户端播放武器发射音效和视觉效果。</p>

本地游戏	网络游戏
玩家1的发射物从武器中射出并前移。	<p>玩家1的发射物从在服务器上的武器中射出并前移。 此时，服务器告知所有客户端复制玩家1发射物发生的移动，因此各客户端上的玩家1发射物便相应移动。</p>
<p>玩家1的发射物撞击玩家2的Pawn。 碰撞将触发摧毁玩家1发射物的函数，对玩家2的Pawn造成伤害，并播放附带音效和视觉效果。玩家2播放画面效果，作为对伤害的响应。</p>	<p>玩家1在服务器上的发射物撞击玩家2的Pawn。 碰撞触发摧毁服务器上玩家1发射物的函数。服务器自动告知所有客户端各自摧毁玩家1发射物副本。碰撞触发告知所有客户端播放附带碰撞音效和视觉效果的函数。玩家2在服务器上的Pawn承受发射物碰撞造成的伤害。玩家2在服务器上的Pawn告知玩家2客户端播放画面效果，作为对伤害的响应。</p>

Actor和ActorComponent

- UPROPERTY 的 Replicated 标记是用来指定一个属性是否需要被复制的，它需要在属性的声明中使用，例如 UPROPERTY(replicated) AActor * Owner;。SetIsReplicated 是用来指定一个组件是否需要被复制的，它是一个函数，可以在运行时调用，例如 SetIsReplicated(true)。
- UPROPERTY 的 Replicated 标记只能用于 Actor 类或其子类的属性，而 SetIsReplicated 只能用于 ActorComponent 类或其子类的实例。
- UPROPERTY 的 Replicated 标记只是表示一个属性可以被复制，但还需要在 GetLifetimeReplicatedProps 函数中添加该属性到 OutLifetimeProps 数组中，才能实现复制功能。而 SetIsReplicated 函数只需要传入一个布尔值，就可以开启或关闭组件的复制功能。13
- actor需要标记bReplicated 为true才可以复制

☆☆☆ Actor复制和属性复制

根据我的搜索结果，属性复制和actor复制的实现方法如下：

- 属性复制需要在actor类中声明要复制的属性，并使用 UPROPERTY(Replicated) 宏来标记它们。然后，需要在actor类中实现 GetLifetimeReplicatedProps 函数，并使用 DOREPLIFETIME 宏来指定每个属性的复制条件。最后，需要在actor类中实现 OnRep_XXX 函数（如果有必要），来处理属性在客户端上发生变化时的逻辑。
- actor复制需要在actor类中设置 bReplicates 和 bAlwaysRelevant 属性为 true，并在构造函数中调用 SetReplicates(true)。然后，需要在actor类中实现 ReplicateSubobjects 函数（如果有必要），并使用 Channel->ReplicateSubobject 来手动复制子对象。或者，可以使用注册子对象列表（Registered Subobjects List）来自动处理子对象的复制，这需要在actor类中设置 bReplicateUsingRegisteredSubObjectList 为 true，并在创建或删除子对象时调用 AddReplicatedSubObject 或 RemoveReplicatedSubObject。

基本复制Actor清单

按照以下步骤，可创建复制Actor：

- 将Actor的复制设置设为True。
- 若复制Actor需要移动，将复制移动（Replicates Movement）设为True。
- 生成或销毁复制Actor时，确保在服务器上执行该操作。
- 设置必须在机器间共享的变量，以便进行复制。这通常适用于以gameplay为基础的变量。
- 尽量使用虚幻引擎的预制移动组件，其已针对复制进行构建。
- 若使用服务器授权模型，需确保玩家可执行的新操作均由服务器函数触发。

网络提示

- 尽可能少用RPC或复制蓝图函数。在合适情况下改用RepNotify。
- 组播函数会导致会话中各连接客户端的额外网络流量，需尤其少用。
- 若能保证非复制函数仅在服务器上执行，则服务器RPC中无需包含纯服务器逻辑。
- 将可靠RPC绑定到玩家输入时需谨慎。玩家可能会快速反复点击按钮，导致可靠RPC队列溢出。应采取措施限制玩家激活此项的频率。
- 若游戏频繁调用RPC或复制函数，如tick时，则应将其设为不可靠。
- 部分函数可重复使用。调用其响应游戏逻辑，然后调用其响应RepNotify，确保客户端和服务器拥有并列执行即可。
- 检查Actor的网络角色可查看其是否为 `ROLE_Authority`。此方法适用于过滤函数中的执行，该函数同时在服务器和客户端上激活。
- 使用C++中的 `IsLocallyControlled` 函数或蓝图中的Is Locally Controlled函数，可检查Pawn是否受本地控制。基于执行是否与拥有客户端相关来过滤函数时，此方法十分拥有。
- 构造期间Pawn可能未被指定控制器，因此避免在构造函数脚本中使用 `IsLocallyControlled`

OnRep函数最好不要是const纯函数（？）会出现一些xiao
